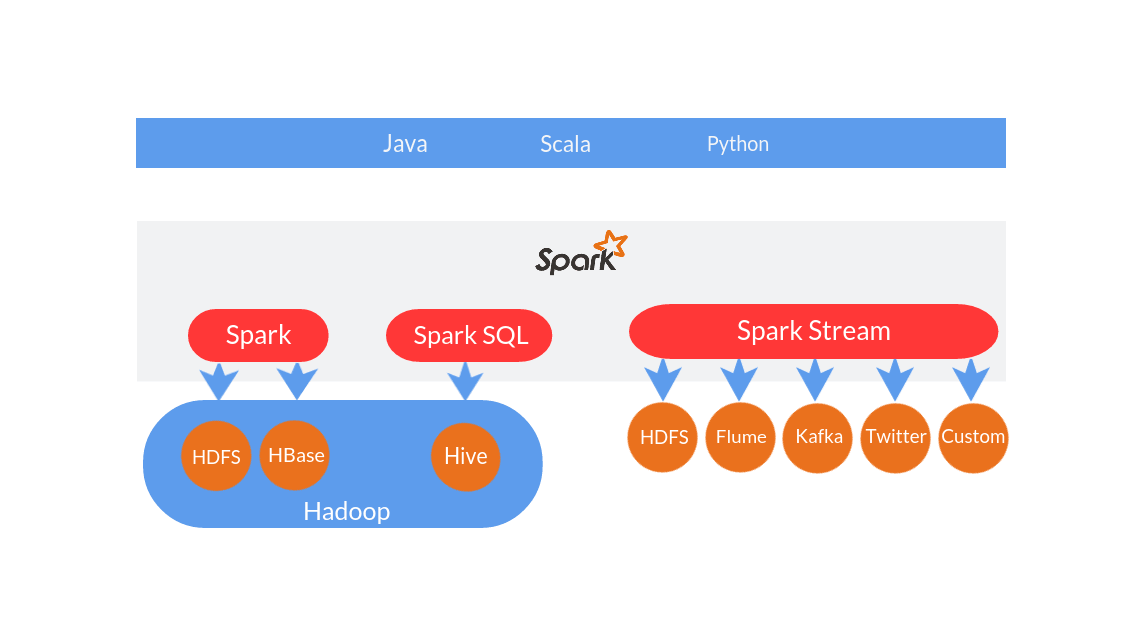
**I**

# *Introduction to Apache Spark*

It is a framework for performing general data analytics on distributed computing cluster like Hadoop.It provides in memory computations for increase speed and data process over mapreduce.It runs on top of existing hadoop cluster and access hadoop data store (HDFS), can also process structured data in Hive and Streaming data from HDFS,Flume,Kafka,Twitter



### Is Apache Spark going to replace Hadoop?

Hadoop is parallel data processing framework that has traditionally been used to run map/reduce jobs. These are long running jobs that take minutes or hours to complete. Spark has designed to run on top of Hadoop and it is an alternative to the traditional batch map/reduce model that can be used for real-time stream data processing and fast interactive queries that finish within seconds. So, Hadoop supports both traditional map/reduce and Spark.

We should look at Hadoop as a general purpose Framework that supports multiple models and We should look at Spark as an alternative to Hadoop MapReduce rather than a replacement to Hadoop.

### Hadoop MapReduce vs. Spark –Which One to Choose?

Spark uses more RAM instead of network and disk I/O its relatively fast as compared to hadoop. But as it uses large RAM it needs a dedicated high end physical machine for producing effective results

It all depends and the variables on which this decision depends keep on changing dynamically with time.

### Difference between Hadoop Mapreduce and Apache Spark

Spark stores data in-memory whereas Hadoop stores data on disk. Hadoop uses replication to achieve fault tolerance whereas Spark uses different data storage model, resilient distributed datasets (RDD), uses a clever way of guaranteeing fault tolerance that minimizes network I/O.

From the Spark academic paper: "RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information to rebuild just that partition." This removes the need for replication to achieve fault tolerance.

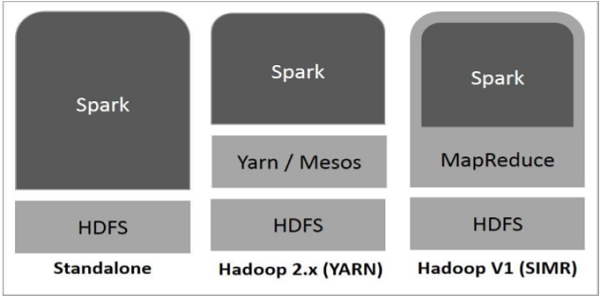
Spark enables applications in Hadoop clusters to run up to 100x faster in memory, and 10x faster even when running on disk. Spark makes it possible by reducing number of read/write to disc. It stores this intermediate processing data in-memory. It uses the concept of an Resilient Distributed Dataset (RDD), which allows it to transparently store data on memory and persist it to disc only it’s needed. This helps to reduce most of the disc read and write – the main time consuming factors – of data processing

### Features of Apache Spark

* **Speed** − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
* **Supports multiple languages** − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
* **Advanced Analytics** − Spark not only supports ‘Map’ and ‘reduce’. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

### Spark Built on Hadoop

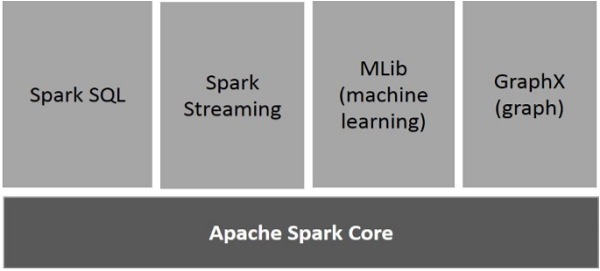
The following diagram shows three ways of how Spark can be built with Hadoop components.



### Spark’s major use cases over Hadoop

* Iterative Algorithms in Machine Learning
* Interactive Data Mining and Data Processing
* Spark is a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive.
* Stream processing: Log processing and Fraud detection in live streams for alerts, aggregates and analysis
* Sensor data processing: Where data is fetched and joined from multiple sources, in-memory dataset really helpful as they are easy and fast to process.

### Components of Spark



#### Apache Spark Core

Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.

#### Spark SQL

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.

#### Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD (Resilient Distributed Datasets) transformations on those mini-batches of data.

#### MLlib (Machine Learning Library)

MLlib is a distributed machine learning framework above Spark because of the distributed memory-based Spark architecture. It is, according to benchmarks, done by the MLlib developers against the Alternating Least Squares (ALS) implementations. Spark MLlib is nine times as fast as the Hadoop disk-based version of Apache Mahout (before Mahout gained a Spark interface).

#### GraphX

GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation that can model the user-defined graphs by using Pregel abstraction API. It also provides an optimized runtime for this abstraction.

**II**

# *Resilient Distributed Dataset (RDD)*

At the core of Spark is the notion of a Resilient Distributed Dataset (RDD), which is an immutable collection of objects that is partitioned and distributed across multiple physical nodes of a YARN cluster and that can be operated in parallel.

There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

Once an RDD is instantiated, you can apply a series of operations. All operations fall into one of two types***: transformations or actions.***

Transformation operations, as the name suggests, create new datasets from an existing RDD and build out the processing Directed Acyclic Graph (DAG) that can then be applied on the partitioned dataset across the YARN cluster.

An Action operation, on the other hand, executes DAG and returns a value.

RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Formally, an RDD is a read-only, partitioned collection of records. RDDs can be created through deterministic operations on either data on stable storage or other RDDs. RDD is a fault-tolerant collection of elements that can be operated on in parallel.

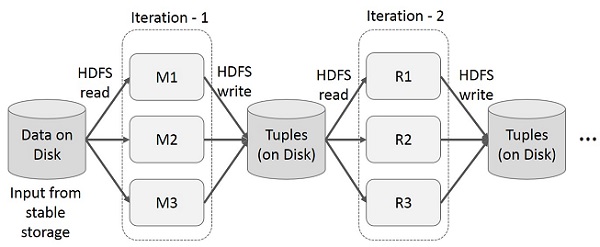
Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations. Let us first discuss how MapReduce operations take place and why they are not so efficient.

### Data Sharing is Slow in MapReduce

In most current frameworks, the only way to reuse data between computations (Ex − between two MapReduce jobs) is to write it to an external stable storage system (Ex − HDFS). Although this framework provides numerous abstractions for accessing a cluster’s computational resources, users still want more.

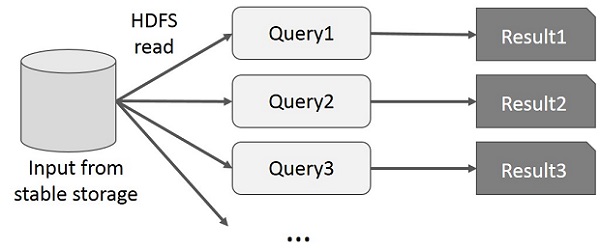
Both **Iterative** and **Interactive** applications require faster data sharing across parallel jobs. Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Regarding storage system, most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

The following illustration explains how the current framework works, while doing the iterative operations on MapReduce. This incurs substantial overheads due to data replication, disk I/O, and serialization, which makes the system slow.



User runs ad-hoc queries on the same subset of data. Each query will do the disk I/O on the stable storage, which can dominates application execution time.

The following illustration explains how the current framework works while doing the interactive queries on MapReduce.



Data sharing is slow in MapReduce due to **replication, serialization**, and **disk IO**. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is **R**esilient **D**istributed **D**atasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

Let us now try to find out how iterative and interactive operations take place in Spark RDD.

The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

**Note** − If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.



This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.



By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also **persist** an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

**III**

# *Core Programming*

Spark Core is the base of the whole project. It provides distributed task dispatching, scheduling, and basic I/O functionalities.

The RDD abstraction is exposed through a language-integrated API. This simplifies programming complexity because the way applications manipulate RDDs is similar to manipulating local collections of data.

### Spark Shell

Spark provides an interactive shell − a powerful tool to analyze data interactively. It is available in either Scala or Python language.

The following command is used to open Spark shell.

$ spark-shell

Let us create a simple RDD from the text file. Use the following command to create a simple RDD.

scala> val inputfile = sc.textFile(“input.txt”)

### RDD Transformations

RDD transformations returns pointer to new RDD and allows you to create dependencies between RDDs. Each RDD in dependency chain (String of Dependencies) has a function for calculating its data and has a pointer (dependency) to its parent RDD.

Spark is lazy, so nothing will be executed unless you call some transformation or action that will trigger job creation and execution. Look at the following snippet of the word-count example.

Therefore, RDD transformation is not a set of data but is a step in a program (might be the only step) telling Spark how to get data and what to do with it.

Given below is a list of RDD transformations.

| # | Transformation | Meaning |
| --- | --- | --- |
| 1 | map(func) | Returns a new distributed dataset, formed by passing each element of the source through a function func. |
| 2 | filter(func) | Returns a new dataset formed by selecting those elements of the source on which func returns true. |
| 3 |  |  |